SHORTEST PATH

This documentation provides a detailed explanation of three scripts that, when executed in sequence, find a specific color of path in an image, preprocess the image by highlighting paths, and then use Dijkstra's algorithm to find and visualize the shortest path between any two nodes. The scripts are:

1. find\_color\_1.py
2. preprocess\_image.py
3. dijktras.py

All input images and csv files are available inside “inputs” folder.

#### Prerequisites

* **Python**: Ensure Python is installed on your system.
* **Libraries**: The following Python libraries are required: opencv-python-headless, numpy, scikit-image, and scipy. Install them using pip install opencv-python-headless numpy scikit-image scipy.

### Script 1: find\_color\_1.py

#### Purpose

This script identifies the average solid color(RGB) in an image.

#### Algorithm

1. **Load Image**: The script loads an image from a specified path(cropped so that only path in visible in the image) using OpenCV.
2. **Convert to RGB**: The loaded image, initially in BGR format, is converted to RGB format.
3. **Calculate Average Color**: It computes the average color of the image by averaging the RGB values of all pixels.
4. **Return Color**: The average color is returned as an RGB tuple.

#### Usage

* **Input**: Path to the image file which contains only path (“path\_color”).
* **Output**: The RGB values of the average color in the image.This RGB value of the path is used to identify the paths in the subsequent preprocessing steps.

### Script 2: preprocess\_image.py

#### Purpose

This script preprocesses an image by converting areas of the path(target RGB) to green and generates a skeletonized version of the image, identifying key nodes and paths as well as numbering the nodes.

#### Algorithm

1. **Load Image**: The image is loaded and converted to RGB format.
2. **Color Conversion**: The script identifies pixels that match a target color (within a specified tolerance) and converts those pixels to green.This results in only the paths being marked as green.
3. **Thresholding and Skeletonization**:
   * The image is converted to grayscale and binarized.
   * A skeletonization process is applied to reduce the binary image to its skeletal form.
4. **Node Identification**: The skeletonized image is analyzed to identify nodes where paths intersect.
5. **Distance Calculation**: The Euclidean distance between nodes is calculated to prepare for pathfinding and node clustering. Nodes too close to each other are clustered(threshold is adjustable).

The output images from these preprocessing steps is available in the “preprocess\_steps” folder.”thin\_image” is an input for the next script.

The transformations applied are in the following order:

Convert all target RGB pixels into green -> process the image(sharpen and blue) -> Convert image into grayscale -> binarize image -> extend path pixels by a predefined value to cover any discontinuities -> skeletonize the image -> identify and number nodes in the image.

#### Usage

* **Input**: Path to the original image and the target RGB color(“image” , RGB value from find\_color\_1 script).
* **Output**:
  + Images from various preprocessing steps(“thin\_image” is an input for the next script).
  + An output image with nodes identified and numbered.

Using the nodes identified in the output image create a csv file containing all the adjacent nodes in the image. This csv file is input for the next code.The csv file of adjacent nodes in available by the name “adjacent\_nodes” inside inputs folder.

### Script 3: dijktras.py

#### Purpose

This script uses Dijkstra's algorithm to find the shortest path between nodes identified in the preprocessed image and visualizes this path on the original image.

#### Algorithm

1. **Load Data**:
   * Load the preprocessed image and the CSV file containing node positions and distances.
   * Load a background image for overlay.
2. **Graph Construction**: Construct a graph where nodes are connected by edges with weights representing distances.
3. **Dijkstra's Algorithm**:
   * Implement Dijkstra's algorithm to find the shortest path between the user-specified start and end nodes.Distance between nodes is calculated as the number of black pixels between two nodes .
   * Highlight this path on the preprocessed image.
4. **Visualization**:
   * Draw the shortest path on the image.
   * Overlay the processed image onto a background image.
5. **Save Results**: Save the image with the highlighted path and the overlaid final image.

#### Usage

* **Input**:
  + Paths to the preprocessed image, the CSV file with node data, and the background image(“thin\_image”, “adjacent nodes” , “image”).
  + Start and end nodes specified by the user.
* **Output**:
  + Image with the shortest path highlighted.
  + Final overlaid image with the highlighted path.
  + The shortest distance and path printed to the console.

### Workflow Summary

1. **Run** find\_color\_1.py: Determine the average color of the target area in the image.
2. **Run** preprocess\_image.py: Preprocess the image to highlight paths and identify nodes, saving the necessary data.
3. **Run** dijktras.py: Use Dijkstra's algorithm to find and visualize the shortest path between nodes in the preprocessed image.

This sequential execution allows for the identification of significant colors, preprocessing of images for pathfinding, and the application of Dijkstra's algorithm to find and visualize the shortest path in the processed image.